

Tutorial: Controlling Access with Servlet Filters

This tutorial describes how the Financial Brokerage Service (FBS) sample application uses *servlet filters* to control end-user access to application features. Introduced in the Java Servlet 2.3 specification, servlet filters enable developers to modify the behavior of servlet-based applications without touching existing code.

This tutorial assumes that you are familiar with the FBS and have installed and configured the required software as described in *About the Financial Brokerage Service*.

Contents

1. Concepts
2. Design
3. Required Software
4. Setup
5. Implementation
6. Resources
7. Feedback





Concepts

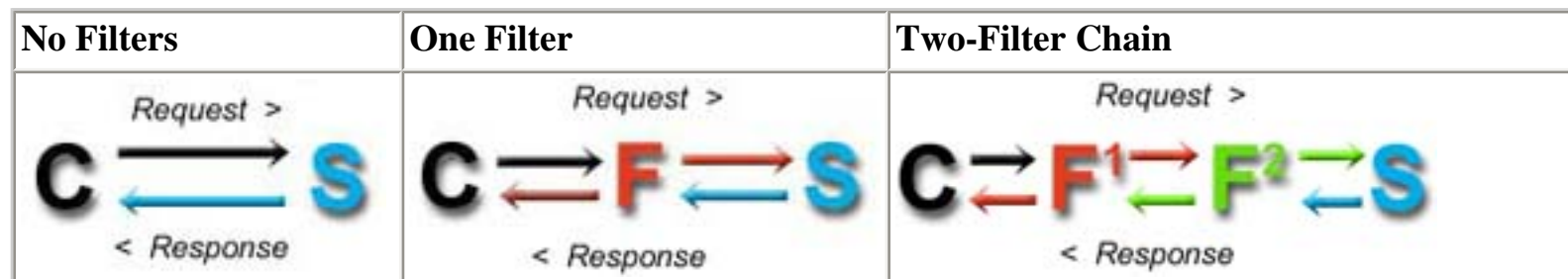
The Java Servlet 2.3 specification introduced the concept of *filters*, components that can intercept and modify requests to and responses from servlets and JSPs (JavaServer Pages). You can extend and enhance an application by adding one or more filters, and you can apply filters individually or in a series called a *filter chain*.

From a developer's point of view, a filter is a Java class that implements the interface defined in `javax.servlet.Filter`. The key method is `doFilter`, which has the following signature:

```
public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
```

As you can see, `doFilter` provides access to the request and response objects, and to the filter chain. This method does the actual filtering, and can be called any number of times. The Filter interface also defines an `init` method, called once before a filter goes into service, and a `destroy` method, called once before the filter is taken out of service.

The following figure shows the request/response flow in three scenarios: an application with a client (C) and a servlet (S) but no filters, the same application using one filter (F), and again using two filters (F1 and F2) in a filter chain. Note that while a filter *can* modify a request or a response object, it doesn't have to. A filter could pass the object along unmodified. Also, you don't need a servlet at the back end to use filters—you can configure an application server (or other container) to apply filters to any request.



In addition to implementing the filter, you must edit the Web application deployment descriptor file (`web.xml`) so the container can find and invoke the filter. Following is a portion of a `web.xml` file that declares a simple filter, the Java class that implements it, and a URL pattern to which the filter will be applied. The filter name, defined by the developer, maps the Java class to the URL pattern. In this example, the filter named `FilterAllRequests` is implemented in `mypackage1.FilterOne.java`, and the container will apply it to all requests it receives (an asterisk is a wildcard in a URL pattern). You can also apply filters to specific directories or resources (files). Other tags are available for declaring filter attributes, but those shown below are essential.

```
...
<filter>
  <filter-name>FilterAllRequests</filter-name>
  <filter-class>mypackage1.FilterOne</filter-class>
</filter>
<filter-mapping>
  <filter-name>FilterAllRequests</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
...
```

Here is XML code that defines another filter. Because of the value specified in the `<url-pattern>` element, the container only applies this filter when it receives a request for a resource in the `/mydocs` directory.

```
...
<filter>
  <filter-name>FilterMyDocs</filter-name>
  <filter-class>mypackage1.FilterTwo</filter-class>
</filter>
<filter-mapping>
  <filter-name>FilterMyDocs</filter-name>
  <url-pattern>/mydocs/*</url-pattern>
</filter-mapping>
...
```

To define a filter chain, put two or more filter declarations into the configuration file and

supply appropriate values for the `<url-pattern>` elements . For example, given the code below, the container will apply `FilterAllRequests` and `FilterMyDocs` when it receives a request for a resource like `http://127.0.0.1:8989/mydocs/foo.html`. When two or more filters apply to the same resource, they are invoked in the order that they appear in the configuration file.

```
...
<filter>
  <filter-name>FilterAllRequests</filter-name>
  <filter-class>mypackage1.FilterOne</filter-class>
</filter>
<filter-mapping>
  <filter-name>FilterAllRequests</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>

<filter>
  <filter-name>FilterMyDocs</filter-name>
  <filter-class>mypackage1.FilterTwo</filter-class>
</filter>
<filter-mapping>
  <filter-name>FilterMyDocs</filter-name>
  <url-pattern>/mydocs/*</url-pattern>
</filter-mapping>
...
```





Design

The Financial Brokerage Service(FBS) controls access to application features by displaying user interfaces based on user roles. For example, Corporate Users can access pages that enable them to set up ESOP (Employee Stock Option Plan) accounts for employees, but Individual Users cannot. The FBS applies a filter to process user requests and return the appropriate pages. Using filters, you can add this kind of access control to a JSP-based application without touching the existing code.

This filter-based access control model is flexible, too, because the data that maps user roles to specific JSPs is stored in an XML file. As a result, you can modify mappings and leave the application in place—there's no need to recompile or redeploy.





Required Software



This tutorial presents several code examples. If you want to study them in context, download and install the FBS source code. If you also want to build and run the FBS, you will need the software listed in the Required Software section of *About the Financial Brokerage Service*.





Setup



If you use Oracle9i JDeveloper as your IDE, add the `J2EE` library to your project. For coding details, see the Implementation section.

To configure your system to build and run the FBS sample application, see the Setup section of *About the Financial Brokerage Service*.





Implementation

Three files are key to the implementation of an access control filter in the FBS:

- `ibfbs/etc/web.xml`
- `ibfbs/src/public_html/xml/Control.xml`
- `ibfbs/src/oracle/otnsamples/ibfbs/control/AccessControlFilter.java`

Here is the part of `web.xml` that defines the class and URL mapping for the access control filter. The filter named `AccessControlFilter` is implemented by the `oracle.otnsamples.ibfbs.control.AccessControlFilter` class. It is invoked each time a user requests a resource from a URI containing the `/controllerservlet` pattern (example: `http://www.mydomain.com/controllerservlet/foo.jsp`).

```
<filter>
  <filter-name>AccessControlFilter</filter-name>
  <filter-class>oracle.otnsamples.ibfbs.control.AccessControlFilter</filter-
class>
</filter>

<filter-mapping>
  <filter-name>AccessControlFilter</filter-name>
  <url-pattern>/controllerservlet</url-pattern>
</filter-mapping>
```

Here are some entries from `ibfbs/src/public_html/xml/Control.xml` that associate events, user roles, and JSPs. An Individual User can buy stock, but only a Corporate User can access the JSP that displays the Corporate Upload screen, and only an Administrator can upload news. However, any user can login.

```
<Event>
  <Name>BUYSTOCK</Name>
  <Class>oracle.otnsamples.ibfbs.trademangement.helper.TradeManagementHelper</Class>
  <Method>buyStock</Method>
  <Screen>jsps/BuyStock.jsp</Screen>
  <Roles>
    <Role>USER</Role>
  </Roles>
</Event>
...
<Event>
  <Name>CORPUPLOAD</Name>
  <Class></Class>
```

```

<Method></Method>
<Screen>jsp/CorporateUpload.jsp</Screen>
<Roles>
  <Role>CORP</Role>
</Roles>
</Event>
...
<Event>
  <Name>CONFIGNEWSUPLOAD</Name>
  <Class>oracle.otnsamples.ibfbs.admin.helper.AdminHelper</Class>
  <Method>configNewsUpload</Method>
  <Screen>jsp/UploadData.jsp</Screen>
  <Roles>
    <Role>ADMIN</Role>
  </Roles>
</Event>
...
<Event>
  <Name>LOGIN</Name>
  <Class>oracle.otnsamples.ibfbs.usermanagement.helper.UserManagementHelper</Class>
  <Method>checkPassword</Method>
  <Screen>jsp/MyHome.jsp</Screen>
  <Roles>
    <Role>DEFAULT</Role>
    <Role>USER</Role>
    <Role>CORP</Role>
    <Role>ADMIN</Role>
  </Roles>
</Event>

```

The FBS reads from the mapping file when the Access Control filter is initialized, and the `AccessControlFilter.doFilter` method handles the filtering chores. After getting the user's role and the URL of the requested page, `doFilter` then checks these values against the mapping data. If the requested page is appropriate for the user's role, the code calls `chain.doFilter` to invoke the page and continue with normal processing. Otherwise, the code calls `request.setAttribute` before `chain.doFilter`, and as a result, the Controller Servlet redirects the user to a login page.

```

public void doFilter(ServletRequest request, ServletResponse response,
                    FilterChain chain)
    throws IOException, ServletException {
    HttpSession session = ((HttpServletRequest) request).getSession();
    String eventName = request.getParameter("EVENTNAME");
    if (eventName != null && urlMap != null ) {
        String role = (String) session.getAttribute("ROLE");
        if (role == null) role = "DEFAULT";
        URLMapping event = (URLMapping) urlMap.get(eventName);
        if ((event != null) && (event.getRoles() != null)
            && (event.getRoles().length > 0)) {

```

```
// New session so not logged in yet. Redirect to login page
if (session.isNew())
    request.setAttribute("EVENTNAME", "FIRSTPAGE");
// If invalid access, redirect to login page
else if (!event.isValidRole(role))
    request.setAttribute("EVENTNAME", "LOGINPAGE");
}
}
else {
    request.setAttribute("EVENTNAME", "FIRSTPAGE");
}
// The privileges are sufficient to invoke this URL, continue normal
// processing of the request
chain.doFilter(request, response);
}
```





Resources

This tutorial is part of a series, *J2EE and EJB 2.0 Techniques*, based on the Financial Brokerage Service (FBS) sample application. Following are links to resources that can help you understand and apply the concepts and techniques presented in this tutorial.

| Resource | URL |
|------------------------------|---|
| OC4J How-To: Servlet Filters | http://otn.oracle.com/tech/java/oc4j/htdocs/how-to-servlet-filter.html |
| Extending Servlet Code | http://otn.oracle.com/sample_code/tech/java/servlets/htdocs/vsmfilters/vsmfiltershome.html |
| The Essentials of Filters | http://java.sun.com/products/servlet/Filters.html |
| Java Servlet Specification | http://java.sun.com/aboutJava/communityprocess/first/jsr053/index.html |





Feedback

If you have questions or comments about this tutorial, you can:

- Post a message in the OTN Sample Code discussion forum. OTN developers and other experts monitor the forum.
- Send email to the author. <mailto:Robert.Hall@oracle.com>

If you have suggestions or ideas for future tutorials, please send email to:

- <mailto:Raghavan.Sarathy@oracle.com>

